Project Two: Multilayer Perceptron

Bradley Reeves
Sam Shissler

Department of Computer Science
Central Washington University
May 6, 2021

Contents

List of Figures

List of Tables

Introduction

This project aims to explore the Multilayer Perceptron neural network model. The MLP is a network consisting of layers of neurons connected by weights, similiar to the Perceptron. Unlike the Perceptron, however, an arbitrary number of layers can be connected. This makes the MLP capable of performing tasks that may not necessarily be linearly separable, such as the XOR problem. This general architecture is shown in Figure 1.1.

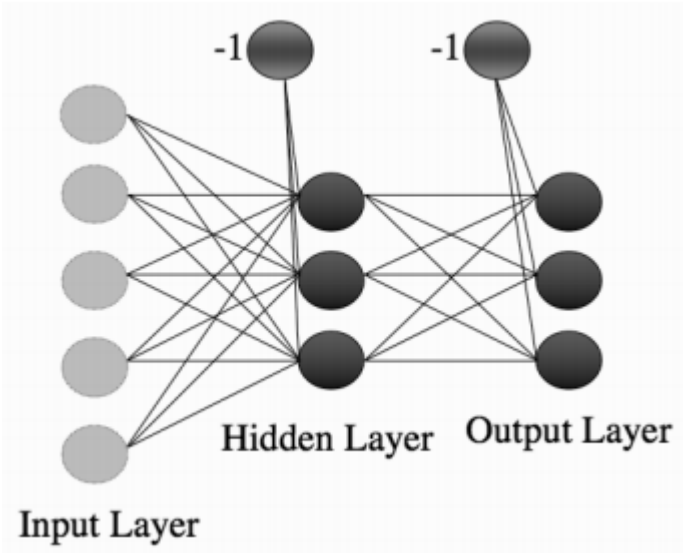


Figure 1.1: The Multilayer Perceptron. [1]

This model works by first creating a set of input nodes based on the input vectors and initializing the model weights with small positive and negative values. Next, the inputs are fed forwards through all layers of the network (Figure 1.2). The forward pass first combines the inputs and weights connecting the inputs to the first hidden layer to decided whether or not the nodes in the hidden layer fire or not. The activation is the sigmoid function given in Equation 1.1 below.

$$\alpha = g(h) = \frac{1}{1 + \exp(-\beta h)} \tag{1.1}$$

The outputs of the hidden layer neurons combined with the weights connecting the hidden layer to the output layer then decide whether or not the output neurons fire.

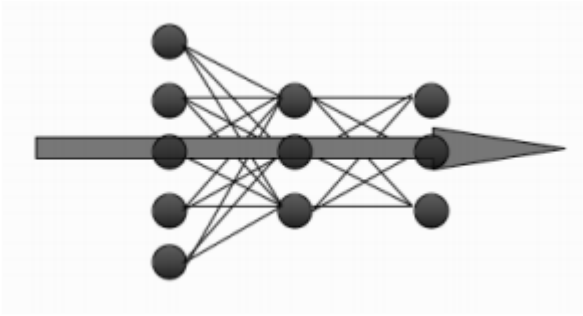


Figure 1.2: The Multilayer Perceptron forward direction. [1]

Next, the error is computed as the sum-of-squares difference (Equation 1.2) between the final outputs and the targets. Finally, this error is propagated backwards through the network by first updating the second layer weights then the first layer weights.

$$E(t, y) = \frac{1}{2} \sum_{k=1}^{N} (y_k - t_k)^2 \tag{1.2}$$

The goal of this project is to approximate Equation 1.3 using the MLP architecture. The number of hidden layers and nodes will be varied in order to investigate how the network's performance is impacted. Four implementations are tested for comparison. These implementations include a custom implementation, a Scikit-Learn implementation, a Keras implementation, and a Weka implementation.

$$f(x, y) = \sin\left(\pi 10x + \frac{10}{1 + y^2}\right) + \ln(x^2 + y^2) \tag{1.3}$$

Dataset

The dataset is relatively simple with two input values and a single target value. The dataset is generated by first initializing x and y vectors to 50 evenly spaced numbers over the interval $[1, 100]$. To get the target value z, $f(x, y)$ is calculated where f is Equation 1.2. A peek at these samples is provided in Table 2.1

	x	y	z
x_1	1.	1.	-0.26577709
x_2	3.02040816	3.02040816	3.90223755
x_2	5.04081633	5.04081633	4.924224
.	.	.	.
.	.	.	.
.	.	.	.
x_n	100.	100.	9.90448745

Table 2.1: Dataset Peek.

Prior to any experimentation, the dataset is randomly shuffled then split into training and test sets using a 60/40 split.

Experimentation

A single experiment was conducted against four MLP implementations. Because the focus is on the number of hidden layers and nodes, the learning rate is held constant at 0.025 and the number of epochs at 500. The experiment consists of a nested for loop where the outer loop iterates over the number of hidden layers (1 and 2) and the inner loop iterates over the number of nodes per hidden layer (1, 2, 3, 4, 5, 10, 15, and 20). Each implementation uses the identity activation function at the output layer. The model architectures look similar to Figure 3.1.

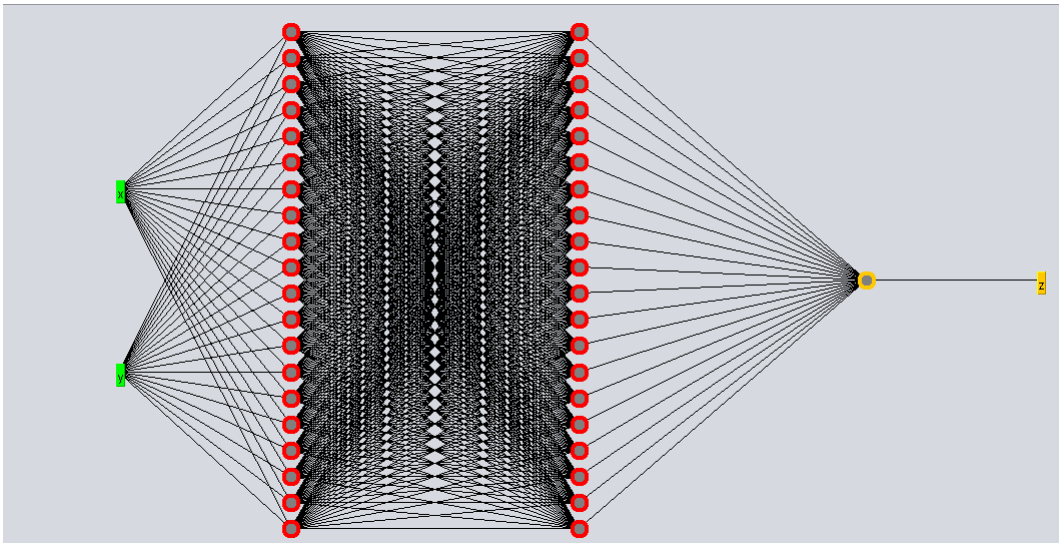


Figure 3.1: Multilayer Perceptron with two hidden layers and 20 nodes per hidden layer.

Analysis

Each implementation was able to approximate the function relatively well. The architecture that performed best for each implementation is given in Table 4.1 below.

Implementation	Hidden Layers	Hidden Layer Nodes	RMSE
Custom	1	15	2.0283
Scikit-Learn	2	4	0.6827
Keras	2	20	0.9773
Weka	1	10	1.3193

Table 4.1: Best results for each implementation.

As shown in Table 4.1, the custom implementation performed the worst while the Scikit-Learn implementation performed the best. For a holistic overview

of the errors from each implementation, see Table 4.2. Note that for each plot, the RMSE is given on the y-axis and the number of nodes per hidden layer is given on the x-axis.

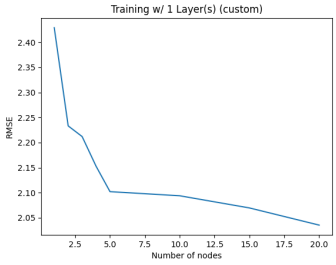
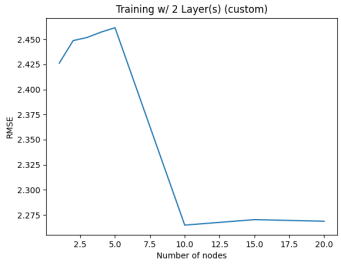
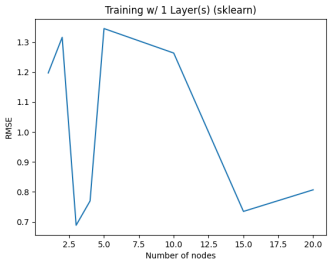
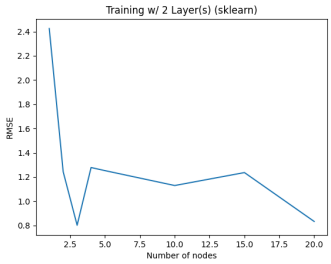
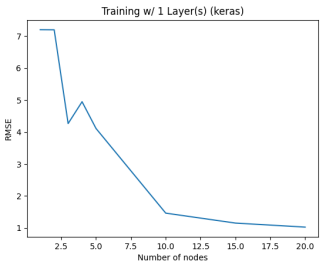
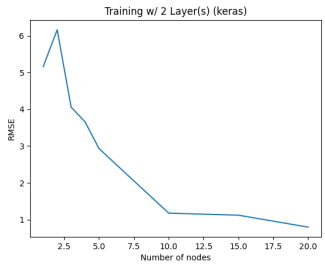
Hidden Layers:	1	2
Custom		
Scikit-Learn		
Keras		

Table 4.2: Experimental Results.

As the results show, the accuracy doesn't necessarily improve as the numbers of hidden layers and nodes per hidden layer increases. In fact, the Keras implementation was the only one that performed best with 2 hidden layers made up of 20 nodes each. An interesting point to make is that nearly the same accuracy could be accomplished with a single hidden layer and an arbitrarily large number of hidden nodes according the Universal Approximation Theorem [1].

In addition to visualizing the error of each model, it's useful to observe the line of best fit. The best fit for each implementation architecture is shown in Figures 4.1, 4.2, and 4.3.

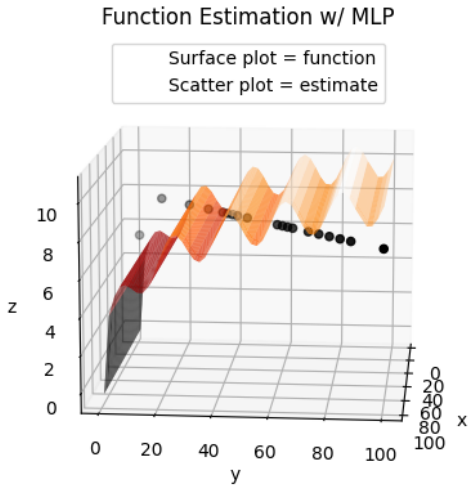


Figure 4.1: Custom Implementation Best Fit.

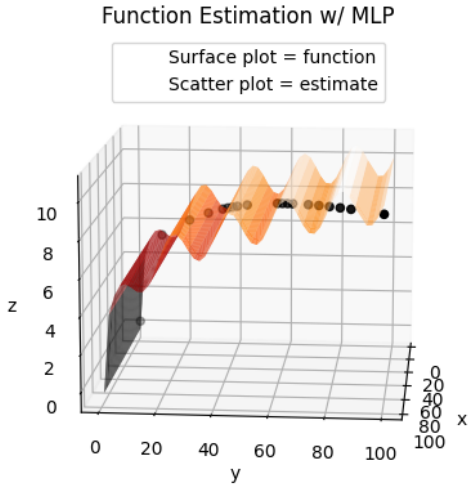


Figure 4.2: Scikit-Learn Implementation Best Fit.

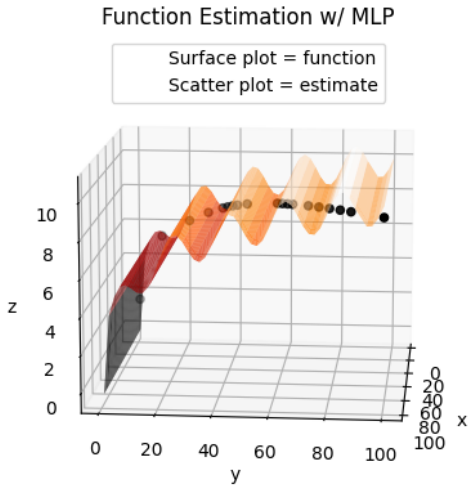


Figure 4.3: Keras implementation Best Fit.

The results in the previous figures agree with our earlier conclusion that the Custom model had the worst performance while the Scikit-Learn model had the best performance. This is most likely because the Scikit-Learn and Keras implementations us packages that are mature and highly optimized. The custom implementation only uses techniques discussed in the MLP chapter of the textbook. The custom implementation could be improved by experimenting with different weight update methods, activation functions, and hyper-parameters. Also, it would be worth experimenting with different weight initialization techniques such as the method described by He et al.

Conclusion

The Multilayer Perceptron is an excellent algorithm for performing regression. The performance of this type model is highly dependent on the implementation used and the overall architecture. This experiment has shown that there is not necessarily a correlation between the accuracy and the number of hidden layers and nodes for all possible architectures. It appears that there is some correlation though, because the accuracy does tend to increase as the number of hidden layers and nodes increase. Varying these parameters

has a large effect on the overall performance of the model and there is not a single solution to always get the best performance. The number of hidden layers and nodes will depend on several factors including the dataset, the task (classification, regression, etc.), and the available resources. Adding thousands of hidden layers and nodes might improve the model, but most researchers do not have time to wait for that type of model to train. If it was guaranteed to improve accuracy, then it might be worth trying, but there is no such guarantee. Additional experiments could help improve our results by varying other hyper-parameters such as the learning rate and number of epochs.

Overall, this experiment was very interesting and serves as a great segue to convolutional neural networks and other machine learning algorithms.

References

[1] S. Marsland, *Machine Learning An Algorithmic Perspective, Second Edition*. CRC Press, 2015.